Imię i nazwisko: **22Z kol2**

Proszę wpisywać odpowiedzi w miejscach na to przeznaczonych

Zadanie 1 (2pkt)

Proszę odpowiedzieć na poniższe pytania:

- Czy (i ewentulnie jak) można mierzyć jakość testu?

- Czy (i ewentualnie jak) można mierzyć "utrzymywalność" (maintability) kodu?

Zadanie 2 (2pkt)

Dana jest funkcja:

```
std::shared_ptr<Item> Service::method(const Other* obj, int x) {
  if (x < this->range().lowerBound)
    throw InvalidArgument(x);
  if ((obj == nullptr) || (obj->isExcluded()))
    return nullptr;
  return (x >= this->range().upperBound) ? Item::Default : std::make_shared<Item>(x);
}
```

Ile przypadków testowych należy napisać, by uzyskać:

- 100% pokrycia linii kodu,
- 100% pokrycia gałęzi kodu (mierzonych narzędziem w rodzaju LCOV).

Podaj wartości argumentów konieczne do uzyskania wspomnianych pokryć.

Zadanie 3 (2pkt)

Co jest nie tak z poniższym kodem?

```
int Service::process(const Request& r, bool ignored, Type t) {
  // sprawdzamy, czy usługa działa
  if (isConfigured() && registry->isEnabled(this) && subservice != nullptr && subservice->isConfigured()
                  && subservice->isActive() && registry->isEnabled(subservice))
  {
    if (!r.header.topic.items.contain(Request::Header::Type::Active))
      return -2;
    auto tmp = r.body.items[3].data;

    if (tmp > 5) {
      tmp = 1685 * itemCount();
      if (ignored)
        throw IncorrectRequest(r);
      switch (t) {
      case Type::Responding: return sendResponse(r, tmp);
      case Type::Blocked: return -3;
      default: return 0;
      }
    } else {
      if (ignored)
        throw IncorrectRequest(r);
      switch (t) {
      case Type::Responding: sendResponse(r, 15 * itemCount() - 2);
      case Type::Blocked: return -3;
      default: return 0;
      }
   }
  }
  else
  {
    return -4;
  }
  return -5;
}
```

Proszę opisać jak należałoby poprawić ten kod (jakie operacje i w jaki sposób wykonane, kod nie jest konieczny).

Uwagi do prowadzącego (K. Grochowskiego):

Imię i nazwisko: **22Z kol2**

Proszę wpisywać odpowiedzi w miejscach na to przeznaczonych

Zadanie 4 - współbieżne wzorce projektowe (2pkt)

Każdy z graczy wykonuje ruchy na planszy, reprezentowanej przez obiekt typu `Board` w niezależnym watku. Niestety klasa ta zawiera błąd, gracz typu `PlayerExtra` powinien wykonywać dwa ruchy jeden po drugim, a obserwujemy sytuacje, w której pomiędzy tymi ruchami tworzony jest ruch gracza typu `PlayerNormal`. Popraw klasę `Board`.

W funkcji 'main' występują obiekty 'ref(x)'. Są to obiekty pomocnicze, które zapobiegają tworzeniu kopii obiektu, więc np. 'thread thrd1(ref(p1));' oznacza, że funkcja wątku obliczeniowego to będzie 'operator()' obiektu 'p1' (a nie jego kopii). Nie jest to kluczowe do znalezienia rozwiązania.

```
class PlayerNormal {
public:
    PlayerNormal(int id, Board& board) : id_(id), board_(board) {}
    void operator()() {
        for(int i=0;i<100;++i) {
            board_.make_move(id_);
            this_thread::sleep_for(chrono::milliseconds(20));
        }
    }
private:
    int id_;
    Board& board_;
};
class PlayerExtra {
public:
    PlayerExtra(int id, Board& board) : id_(id), board_(board) {}
    void operator()() {
        for(int i=0;i<100;++i) {
            board_.make_two_moves(id_);
            this_thread::sleep_for(chrono::milliseconds(20));
        }
    }
private:
    int id_;
    Board& board_;
};

int main() {
    Board board;
    PlayerNormal player1(1,board);
    PlayerExtra player2(2,board);
    thread thrd1( ref(player1) );
    thread thrd2( ref(player2) );
    thrd1.join();
    thrd2.join();
    return 0;
}
```

```
class Board {
public:
 void make_move(int player_id) {
     lock_guard<mutex> guard(m_);
     history_.push_back(player_id);
 }

 void make_two_moves(int player_id) {
     make_move(player_id);
     make_move(player_id);
 }
private:
    mutex m_;
    vector<int> history_;
};
```

Uwagi do prowadzącego (R. Nowaka):

Zadanie 5 - współbieżne wzorce projektowe (2pkt)

Przedstawiona aplikacja zachowuje się niepoprawnie. Popraw funkcję wykonywaną w wątkach użytkownika.

W funkcji 'main' występują obiekty 'ref(x)', opisane w Zadaniu 4.

```
static const int NUM = 10;

/** klasa wejscia - wyjscia do kolejki znakow */
struct Queue {
    list<int> dane_;
    mutex mutex_;
};

int main() {
    Queue q1, q2, q3;
    q1.dane_.push_back(100);
    Producent p1(1, q1, q2);
    Producent p2(2, q2, q3);
    Producent p3(3, q3, q1);
    thread thrd1( ref(p1) );
    thread thrd2( ref(p2) );
    thread thrd3( ref(p3) );
    thrd1.join();
    thrd2.join();
    thrd3.join();
    return 0;
}
```

```
struct Producent {

    Producent(int id, Queue& in, Queue& out) : id_(id), in_(in), out_(out) {}

    void operator()() {
        for(int num = 0; num < NUM; ) {
            lock_guard<mutex> lock_in(in_.mutex_);
            lock_guard<mutex> lock_out(out_.mutex_);
            if(!in_.dane_.empty() ) {
                int x = in_.dane_.front();
                in_.dane_.pop_front();

                //tutaj przetwarza dane lokalnie

                out_.dane_.push_back(x+1);
                ++num;
            }
        }
    }
    int id_;
    Queue& in_;
    Queue& out_;
};
```